

```
1
2 #include <math.h>
3 #include <string.h>
4
5 #include "tiff.h"
6 #include "allocate.h"
7 #include "randlib.h"
8 #include "typeutil.h"
9 #include "defs.h"          // header file that contains the necessary
                             function prototypes
10
11 void error(char *name);
12
13 int main (int argc, char **argv)
14 {
15     FILE *fp;
16     struct TIFF_img input_img, output_img;
17     unsigned int** filtered;
18     int32_t i,j;
19
20     if ( argc != 2 ) error( argv[0] );
21
22     /* open image file */
23     if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
24         fprintf ( stderr, "cannot open file %s\n", argv[1] );
25         exit ( 1 );
26     }
27
28     /* read image */
29     if ( read_TIFF ( fp, &input_img ) ) {
30         fprintf ( stderr, "error reading file %s\n", argv[1] );
31         exit ( 1 );
32     }
33
34     /* close image file */
35     fclose ( fp );
36
37     /* check the type of image data */
38     if ( input_img.TIFF_type != 'g' ) {
39         fprintf ( stderr, "error: image must be grayscale\n" );
40         exit ( 1 );
41     }
42
43     /* set up structure for output achromatic image */
44     /* to allocate a full color image use type 'c' */
45     get_TIFF ( &output_img, input_img.height, input_img.width, 'g' );
46
47     /* call filterImage function which loops through each */
48     /* pixel and applies weighted median filtering */
```

```
49     filtered = filterImage(input_img);
50
51     /* assign each element of filtered array to the */
52     /* monochrome data of the output image TIFF object */
53     for (i = 0; i < input_img.height; i++) {
54         for (j = 0; j < input_img.width; j++) {
55             output_img.mono[i][j] = filtered[i][j];
56         }
57     }
58
59     /* free data from filtered variable */
60     free_img((void*)filtered);
61
62     /* open green image file */
63     if ( ( fp = fopen ( "filtered_output.tif", "wb" ) ) == NULL ) {
64         fprintf ( stderr, "cannot open file green.tif\n");
65         exit ( 1 );
66     }
67
68     /* write green image */
69     if ( write_TIFF ( fp, &output_img ) ) {
70         fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
71         exit ( 1 );
72     }
73
74     /* close green image file */
75     fclose ( fp );
76
77     /* de-allocate space which was used for the images */
78     free_TIFF ( &(input_img) );
79     free_TIFF ( &(output_img) );
80
81     return(0);
82 }
83
84 void error(char *name)
85 {
86     printf("usage:  %s  image.tiff \n\n",name);
87     printf("this program reads in a 24-bit color TIFF image.\n");
88     printf("It then horizontally filters the green component, adds noise, \n");
89     printf("and writes out the result as an 8-bit image\n");
90     printf("with the name 'green.tiff'.\n");
91     printf("It also generates an 8-bit color image,\n");
92     printf("that swaps red and green components from the input image");
93     exit(1);
94 }
95
96 unsigned int calculateWMAtLocation(struct TIFF_img input, int xpos, int
```

```
    ypos) {
197     /* initialize necessary variables & arrays */
198     int temp, i, j, k, istar, weightedMedian, sum1, sum2;
199     unsigned int input_pixels[25];
200     unsigned int weightingFactors[25] = { 1, 1, 1, 1, 1,
201     1, 2, 2, 2, 1,
202     1, 2, 2, 2, 1,
203     1, 2, 2, 2, 1,
204     1, 1, 1, 1, 1 };
205
206     /* populating input_pixels array from input TIFF object */
207     k = 0;
208     for (i = xpos - 2; i < xpos + 3; i++) {
209         for (j = ypos - 2; j < ypos + 3; j++) {
210             input_pixels[k] = input.mono[i][j];
211             k++;
212         }
213     }
214
215     /* apply brief sorting algorithm for both input pixels */
216     /* and corresponding weighting factors */
217     for (i = 0; i < 25; i++) {
218         for (j = i + 1; j < 25; j++) {
219             if (input_pixels[i] < input_pixels[j]) {
220                 temp = input_pixels[i];
221                 input_pixels[i] = input_pixels[j];
222                 input_pixels[j] = temp;
223
224                 temp = weightingFactors[i];
225                 weightingFactors[i] = weightingFactors[j];
226                 weightingFactors[j] = temp;
227             }
228         }
229     }
230
231     /* initialize summing variables */
232     sum1 = 0;
233     sum2 = 0;
234     for (i = 0; i < 25; i++) {
235         sum2 += sum2 + weightingFactors[i];
236     }
237
238     /* determining value for weighted median by incrementing */
239     /* i* and using Equation 7 from the assignment handout */
240     istar = 0;
241     while (sum1 < sum2) {
242         sum1 = 0;
243         sum2 = 0;
244         for (i = 0; i <= istar; i++) {
```

```
145         sum1 += sum1 + weightingFactors[i];
146     }
147     for (i = istar + 1; i < 25; i++) {
148         sum2 += sum2 + weightingFactors[i];
149     }
150
151     if (sum1 >= sum2) {
152         weightedMedian = input_pixels[istar];
153     }
154     else {
155         istar++;
156     }
157 }
158 return weightedMedian;
159 }
160
161 unsigned int** filterImage(struct TIFF_img input) {
162     /* initializing looping variables and 2D storage array */
163     int i, j;
164     unsigned int** filtered = (unsigned int**)get_img(input.width,
165         input.height, sizeof(unsigned int));
166
167     /* loop through each pixel of input image and apply weighted */
168     /* median filtering if proper window can be constructed */
169     for (i = 0; i < input.height; i++) {
170         for (j = 0; j < input.width; j++) {
171             if (i >= 2 && i < input.height - 2 && j >= 2 && j <
172                 input.width - 2) {
173                 filtered[i][j] = calculateWMAAtLocation(input, i, j);
174             }
175             else {
176                 filtered[i][j] = input.mono[i][j];
177             }
178         }
179     }
180     return filtered;
181 }
```